The path to an increase in MINOS computing

Ryan Patterson

July 28, 2008

1 The situation

At present MINOS analyzers have access to forty-one local computing cores through Condor, thirty-six FNALU batch cores, sixty-four MINOS grid cores (with AFS access), and a few hundred general purpose grid cores (without AFS access).[1] Discussions at Ely concluded that a ten-fold increase in available computing power is needed if we are to relieve current analysis bottlenecks in any consequential way. FNAL already has this level of computing power available. Figure 1 shows that of the 13k cores available on FermiGrid, an average of 3k sit idle. (This number will vary with other collaborations' usage.) We are encouraged to consume these idle cycles, but there are hurdles we must first clear. This document outlines the issues as I currently understand them and concludes with a call for volunteers to get us through them.

2 What needs doing

The necessary steps can be grouped into the three quasi-independent areas listed below. One volunteer per area (maybe two for the last area) working nearly full time for 4-12 weeks is likely the optimal arrangement. Once folks are identified, I propose we hold a weekly phone meeting with MINOS computing experts to ensure that things stay on track.

[1] The last set has a restricted user list, in part because of the lack of AFS access and the cumbersome registration process.

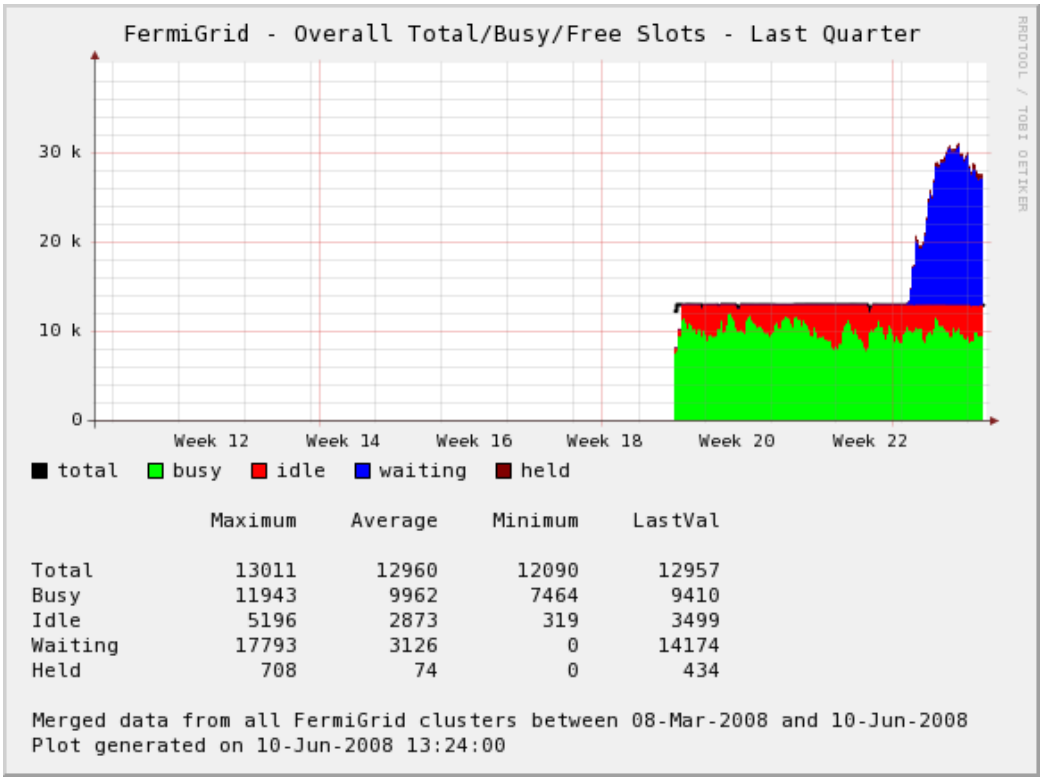

Figure 1: FermiGrid statistics from sometime this spring.

2.1 dCache

Much of MINOS processing involves pulling files from dCache, the caching system that sits between the end user and Enstore (the FNAL tape system). Two main upgrades are needed before dCache can handle thousands of simultaneous jobs.

First, the number of files open in dCache is limited by the number of dCache "doors". There are just two unsecured doors (ports 24125 and 24136), each of which can support a few hundred files. The dCache administrators need to configure several more doors. Our framework code needs to know which doors are available and should distribute the load between doors.

Second, end users can access dCache content in two distinct ways: (1) copy files from dCache to a local area and then run over the local copy, or (2) have the job (usually `loon`) establish a connection to dCache directly and

process over the streamed content. The latter method ties up a dCache slot for the duration of the job, making the number of dCache slots (rather than the number of cores) the limiting factor. To the extent that dCache I/O time is small compared to CPU time, the solution is to use the "copy, then run" sequence always.

Thus, one needs to modify the `loon` dCache code such that the end user *thinks* the files are streaming when in fact they are being copied locally first. As part of this, one should contemplate typical file sizes and temporary storage areas. (For example, is `/tmp` large enough, or must we set aside something else? What about cleanup? Will a multi-file job have to progress in steps: copy, run, copy, run, etc.?)

2.2 Database

It is likely that a single database server cannot accommodate the number of connections needed by thousands of jobs. Step one is to determine how many simultaneous "typical" connections the server can handle. One approach is to install a clone database server on an available machine (perhaps one of the minosXX machines) and test to failure under various conditions.

Assuming the capacity of a single server would indeed be limiting, the next step would be to design a multi-server load-sharing solution. One would implement a turnkey procedure for creating and maintaining database clones. Once the clones are ready, job setup scripts (e.g., setup_minos) would pick a clone at random when setting database environment variables. (Load-dependent database assignment would be better, but random is easier to implement and should be okay at first.) Clones could be added anytime the database becomes the limiting factor (e.g., when new grid cores are made available or when more taxing code is written.)

Along with scaling server capacity, it may become necessary to review database connection hygiene in the MINOS code. Database connections should be held open for as long as it takes to obtain the necessary content, and no longer. Relatedly, multiple sequential database calls should, when possible, be converted into a single up-front call. (`loon` has some ability to deal with the latter case, but it may need updating as analyses progress.) It would also be nice to have a system that monitors the health of the database servers and looks for problem jobs/connections.

2.3 Everything else

This area would likely benefit from more than one full time worker, as the tasks are smaller but manifold. A sampling (the list will likely expand) –

- *Getting jobs to other grid areas.* A system called glideinWMS is used to get each job submitted from `minos25` over to the farm node that actually runs the job. This system may need configuring if we are to submit to other FermiGrid nodes.

- *Condor user interface.* Regardless of the behind-the-scenes picture, end users should have a single job submission paradigm. Ideally, this would be reduced to something as simple as "`minos_jobsub` myexec <myargs>". Condor access on all MINOS machines would also be helpful (rather than just on `minos25`).

- *Executables.* Without AFS access[2], `minossoft` releases (etc.) are not available at the farm nodes. This requires a solution. A promising tool used by CDF and BaBar is *Parrot*, which takes over file I/O transparently, shuttling data through available channels (*e.g.*, HTTP). Whatever the scheme, it needs to be integrated into the MINOS framework.

- *Parrot verification.* If *Parrot* is the solution to the above issue, several of its features (*e.g.*, local sharing of the file cache) must be tested on realistic analysis jobs.

- *Queues, priorities.* Further down the line, we may have need for multiple MINOS queues (*e.g*, test, short, long) and a more complex priority system.

- *Job utilities.* One cannot expect thousands of jobs to run without failures. Tools for seamless job recovery will be useful.

- *Administrative.* As the system comes to life, we will want good monitoring, accounting, and testing tools (in most cases already available, if unfamiliar.) We would also benefit from an Integration Condor system on which to test new Condor releases and configuration changes without affecting end users.

[2] AFS will not scale to the desired level.

3 Want to help?

The effort outlined here is large but certainly doable. What is needed from the collaboration:

- A volunteer for dCache issues.
- A volunteer for database issues.
- An additional volunteer (along with me) for everything else.

Volunteers are hereby requested. Since experts tend to have many constraints on their time, I foresee mostly novice volunteers (who will interact regularly with experts). However, if an expert feels (s)he can knock off a task or two, please do!

Email me if you'd like to take on one of these tasks.

4 Design suggestions

To everyone else: Now's the best time to make requests for the future of MINOS computing. Is there a specific interface quirk you'd like to avoid or a feature you think we need? Let me know what you'd like the system to do, and we can try to make it happen.

5 Closing note

Many thanks to Art Kreymer for laying out the MINOS computing situation for me. This document is mostly my retelling of what I learned from him.